Contenido

Manejo de errores.

En este apartado estudiaremos cómo hacemos el control de los errores producidos en tiempo de ejecución dentro de una rutina.

Por defecto, cualquier error que se produce en una rutina aborta la ejecución de la rutina. Si queremos capturar los errores y recuperarnos habrá que hacerlo mediante el uso de un blog HANDLER.

Un HANDLER nos permite capturar una determinada condición (error) y lleva asociado un código que se ejecuta si la condición se da.

Los utilizaremos principalmente para capturar errores en transacciones y cursores.

Nota: También podríamos dar un nombre personalizado a determinado error y después utilizar el nombre en un HANDLER con DECLARE CONDITION (Manual de referencia punto: 19.2.10.1. Condiciones DECLARO).

DECLARAR ... MANEJADOR

```
DECLARAR handler_action MANEJADOR
  PARA valor_de_condición [, valor_de_condición] ...
  declaración

acción del controlador:
  CONTINUAR
  | SALIDA

valor_de_condición:
  código de error de mysql
  | SQLSTATE [VALOR] valor_sqlstate
  | nombre_de_la_condición
  | ADVERTENCIA SQL
  | NO ENCONTRADO
  | SQLEXCEPTION
```

Acción del controlador

Las acciones posibles que podemos seleccionar como handler_action son:

- CONTINUE: La ejecución del programa sigue después de ejecutar el código asociado al HANDLER .

- EXIT: La ejecución del programa termina después de ejecutar el código asociado a

  EL DISTRIBUIDOR..

Las declaraciones de los HANDLERS siempre son las últimas. DECLARO de las variables locales y cursores.

Normalmente, uno usaremos HANDLER EXIT cuando nuestro objetivo sea detectar errores (en las transacciones, por ejemplo).

Para controlar la iteración de un bucle como en el caso de los CURSORS, el HANDLER debe ser CONTINUAR.

Ejemplo indicando el número de error de MySQL:

En este ejemplo estamos declarando un handler que se ejecutará cuando se produzca el error 1051 de MySQL, que ocurre cuando se intenta acceder a una tabla que no existe en la base de datos. En este caso la acción del handler es CONTINUE lo que significa que después de ejecutar las instrucciones especificadas en el cuerpo del handler el procedimiento almacenado continuará su ejecución.

```
DECLARAR CONTROLADOR CONTINUO PARA 1051
COMENZAR
  -- cuerpo del manipulador
FIN;
```

Ejemplo de un SQLSTATE:

También podemos indicar el valor de la variable SQLSTATE. Por ejemplo, cuando se intenta acceder a una tabla que no existe en la base de datos, el valor de la variable

SQLSTATE y 42S02.

```
DECLARAR MANEJADOR DE SALIDA PARA SQLSTATE '42S02'
COMENZAR
  -- cuerpo del manipulador
FIN;
```

Diferentes códigos de error pueden dar lugar a un mismo SQLSTATE. Por ejemplo: el SQLSTATE '23000' se produce por diferentes errores (con códigos distintos) relacionados con la violación de las restricciones de mesa principalmente.

• Error: 1022 ESTADO SQL: 23000 (ER_DUP_KEY)
Mensaje: No puedo escribir, clave duplicada en la tabla '%s'

• Error: 1048 ESTADO SQL: 23000 (ER_BAD_NULL_ERROR)
Mensaje: La columna '%s' no puede ser nula

• Error: 1052 ESTADO SQL: 23000 (ER_NON_UNIQ_ERROR)
Mensaje: La columna: '%s' en %s es ambigua

• Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY)
Mensaje: Entrada duplicada '%s' para la clave %d

• Error: 1169 ESTADO SQL: 23000 (ER_DUP_UNIQUE)
Mensaje: No puedo escribir, debido al único constraint, para tabla '%

• Error: 1216 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW)
Mensaje: No puede adicionar una línea hijo: falla de clave extranjera constraint

• Error: 1217 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED)
Mensaje: No puede deletar una línea padre: falla de clave extranjera constraint

• Error: 1451 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED_2)
Mensaje: No se puede eliminar ni actualizar una fila principal: falla una restricción de clave externa (%s) • Error: 1452 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW_2)
Mensaje: No se puede agregar ni actualizar una fila secundaria: falla una restricción de clave externa (%s)

(Manual de referencia Capítulo 26. Manejo de errores en MySQL)

Ejemplo de un SQLWARNING:

Es equivalente a indicar todos los valores de SQLSTATE que comienzan con 01.

```
DECLARAR CONTROLADOR CONTINUO PARA SQLWARNING
COMENZAR
  -- cuerpo del manipulador
FIN;
```

Ejemplo para NO ENCONTRADO:

Es equivalente a indicar todos los valores de SQLSTATE que comienzan con 02 . usaremos cuando estemos trabajando con cursores para controlar qué ocurre cuando un cursor alcanza el final del fecha siete. Si no hay más filas disponibles en el cursor, entonces ocurre una condición de NO FECHA con un valor de SQLSTATE igual a 02000. Para detectar esta condición podemos usar NOT FOUND o SQLSTATE '02000' en un handler para controlarlo.

```
DECLARAR CONTROLADOR CONTINUO PARA NO ENCONTRADO
COMENZAR
  -- cuerpo del manipulador
FIN;
```

Ejemplo de una SQLEXCEPTION::

Es equivalente a indicar todos los valores de SQLSTATE que NO comienzan por 00, 01 y 02.

```
DECLARAR CONTROLADOR CONTINUO PARA SQLEXCEPTION

COMIENZA -- cuerpo del manejador
FIN;
```

Ejemplo 1 - DECLARAR MANEJADOR CONTINUO

```
-- Paso 1
Prueba DROP DATABASE IF EXISTS ;
Prueba CREAR BASE DE DATOS;
Prueba de USO ;

-- Paso 2
CREAR TABLA test.t(s1 INT, CLAVE PRIMARIA (s1));

-- Paso 3
DELIMITADOR $$
CREAR PROCEDIMIENTO handlerdemo()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x = 1; SET @x = 1; INSERT INTO
  test.t VALUES (1);
  SET @x = 2; INSERT INTO test.t VALUES (1); --
  Este 2n INSERT
  producirá el SQLSTATE '23000'

-- Al ser CONTINUE, se ejecutará la última sentencia

  ESTABLECER @x = 3;
FIN
$$

DELIMITADOR ;
manejador de llamadasdemo ();
SELECCIONAR @x;
```

El segundo INSERT producirá un error (Error Code: 1062. Duplicate entry '1' for key 'PRIMARY') por violación de Clave Primaria el cual se corresponde con el SQLSTATE '23000' y entra dentro de los estados agrupados en SQLEXCEPTION. Se podía haber detectado de cualquiera de las formas.

Cuando se produce el error, se ejecutan las acciones de HANDLER; en nuestro caso:

```
ESTABLECER @x = 1;
```

Como el HANDLER es CONTINUE sigue la ejecución. La siguiente sentencia es: SIETE @x

= 3;

Por eso, la función devuelve el valor 3.

Ejemplo 2 - DECLARAR MANEJADOR DE SALIDA

```
-- Paso 1
Prueba DROP DATABASE IF EXISTS ;
Prueba CREAR BASE DE DATOS;
Prueba de USO ;

-- Pas 2
CREAR TABLA test.t (s1 INT, CLAVE PRIMARIA (s1));

-- Pas 3
DELIMITER $$ CREAR
PROCEDIMIENTO handlerdemo ()
BEGIN
  DECLARAR MANEJADOR DE SALIDA PARA SQLSTATE '23000' SET @x = 1; SET @x = 1;
  INSERTAR EN
  test.t VALORES (1); SET @x = 2; INSERTAR EN
  test.t VALORES
  (1); -- Este 2n INSERT producirá el SQLSTATE
'23000'
-- Al ser EXIT se ejecuta el código HANDLER y finaliza el PROCEDURE

  ESTABLECER @x = 3;
FIN
$$

DELIMITADOR ;
manejador de llamadasdemo ();
SELECCIONAR @x;
```

Cuando se produce el error, se ejecutan las acciones de HANDLER; en nuestro caso:

ESTABLECER @x = 1;

Como el HANDLER es EXIT, sale del PROCEDURE

Por eso, la función devuelve el valor 1.

Cómo realizar transacciones

Cuando un PROCEDIMIENTO realiza cambios en los datos (INSERTAR, ELIMINAR o ACTUALIZAR:

afectan al estado de la BD), resulta de especial importancia el uso de Transacciones.

Las Transacciones es una herramienta que tienen las BD para garantizar la integridad de las

datos.

Una transacción es una serie de sentencias de manipulación de datos (DML) que constituyen una unidad. Si fallara alguna de las sentencias de la transacción, la rutina debería ser capaz de deshacer todos los cambios hechos hasta ese momento por a dejar las tablas tal y como estaban originalmente.

Por defecto, MySQL se ejecuta con el modo autocommit activado. Esto significa que cuando ejecute un comando que actualice (modifique) una tabla, MySQL almacena la actualización en disco.

Con START TRANSACTION, autocommit permanece deshabilitado hasta el final de la transacción con COMMIT o ROLLBACK.

- Si se ejecuta COMMIT, los cambios producidos en la BD en la transacción se guardarán en disco.
- Si se ejecuta ROLLBACK, la BD volverá al estado que tenía antes de empezar la transacción (antes de START TRANSACTION)

Podemos utilizar el manejo de errores para decidir si hacemos ROLLBACK de una transacción.

En el siguiente ejemplo capturaremos los errores que se produzcan de tipos SQLEXCEPTION y SQLWARNING.

Ejemplo:

```
DELIMITADOR $$
CREAR PROCEDIMIENTO transaccion_en_mysql()
COMENZAR
  DECLARAR MANEJADOR DE SALIDA PARA SQLEXCEPTION
  COMENZAR
  -- ERROR
  RETROCEDER;
  FIN;

  DECLARAR MANEJADOR DE SALIDA PARA SQLWARNING
  COMENZAR
  -- ADVERTENCIA
  RETROCEDER;
  FIN;

  INICIAR TRANSACCIÓN;
  -- Sentencias SQL
  COMPROMETERSE;
FIN
$$
```

En lugar de tener un HANDLER para cada tipo de error, si haremos lo mismo en los dos casos, podemos tener un común.

```
DELIMITADOR $$
CREAR PROCEDIMIENTO transaccion_en_mysql()
COMENZAR
  DECLARAR MANEJADOR DE SALIDA PARA SQLEXCEPTION, SQLWARNING
  COMENZAR
  -- ERROR, ADVERTENCIA
  RETROCEDER;
  FIN;

  INICIAR TRANSACCIÓN;
  -- Sentencias SQL
  COMPROMETERSE;
FIN
$$
```

Ejemplo: Si quisiéramos borrar una fila de Pedidos, también deberíamos borrar todas las filas de DetallePedidos relacionadas. Encapsula los dos DELETE dentro de una transacción

```
delimitador // crear
procedimiento delPedido( codi int unsigned) begin


        declarar el controlador de salida para la reversión de sqlexception ;

        INICIAR TRANSACCIÓN;
                ELIMINAR DE DetallesDePedido DONDE CódigoDePedido=codi;
                ELIMINAR DE Pedidos DONDE OrderCode=codi;
        COMPROMETERSE;

fin //

delimitador;
LLAMA delPedido(4);
```

Cursores

Los cursores nos permiten almacenar un conjunto de filas resultado de una consulta en una estructura de datos que podemos ir recorriendo de forma secuencial.

Los cursores en MySql tienen las siguientes propiedades:

- Asensible: El servidor puede o no realizar una copia de su tabla de resultados.
- Read only: son de sólo lectura.

- Nonscrollable: sólo pueden ser recorridos en una dirección y no podemos saltar-nos files.

> Cuando declaramos un cursor dentro de un procedimiento almacenado debe aparecer antes de las declaraciones de los manejadores de errores (HANDLER) y después de la declaración de variables locales.

Operaciones con cursores

Las operaciones que podemos realizar con los cursores son las siguientes:

DECLARAR

El primer paso que debemos dar para trabajar con cursores es declararlo. La sintaxis para declarar un cursor es:

```
DECLARAR cursor_name CURSOR PARA select_statement
```

ABIERTO

Una vez que hemos declarado un cursor debemos abrirlo con OPEN.

```
ABRIR cursor_name
```

BUSCAR

Una vez que el cursor está abierto podemos ir obteniendo cada una de las filas con FETCH. La sintaxis es la siguiente:

```
OBTENGA [[SIGUIENTE] DE] nombre_cursor EN nombre_var [, nombre_var] ...
```

> Cuando se está recorriendo un cursor y no quedan filas por recorrer, se lanza el error NOT FOUND que se corresponde con el valor SQLSTATE '02000'.

Para trabajar con cursores, se puede optar por controlar el bucle con un handler (declarar un handler para manejar este error) u obtener y guardar en una variable local el número de filas almacenadas en el cursor antes de realizar BUSCAR...

```
DECLARAR CONTROLADOR CONTINUO PARA NO ENCONTRADO ...
```

CERCA

Cuando hemos terminado de trabajar con un cursor debemos cerrarlo. Es recomendable.

```
CERRAR cursor_name
```

Ejemplo 1: Utilizando la variable cuántos para controlar el bucle

En la BD Jardinería. Función que reciba una Ciudad como parámetro y nos devuelva en una cadena de los nombres de los clientes de esa ciudad separados por comas, utilizando cursores.

```
DELIMITADOR $$
CREATE FUNCTION LlistaClientes (PCiu VARCHAR(20))
DEVUELVE TEXTO
COMENZAR
  DECLARE cuántos INT DEFAULT 0;
  DECLARAR Nombre VARCHAR(50) predeterminado '';
  DECLARAR Resultado TEXTO predeterminado '';

  /*es declara el cursor para recorrer la consulta que obtiene los clientes*/
  DECLARAR cursor1 CURSOR PARA SELECCIONAR NombreCliente FROM Clientes
                                        DONDE Ciudad=PCiu;

  /*se obtiene el número de registros para utilizarlo en el WHILE*/
  SELECT count(*) INTO cuántos
  DE Clientes
  DONDE Ciudad=PCiu;

  /*se abre el cursor*/
  ABIERTO cursor1;
  SI quants>=1 entonces
          OBTENGA el cursor1 EN Nom;
          /*En el primero, no hay que meter como delante */
          SET Resultado =Nom;
  FIN SI;

  /*se van concatenando los nombres mientras no llegamos al final del cursor*/
  MIENTRAS quants>1 HACER
          OBTENGA el cursor1 EN Nom;
          SET Resultado = CONCAT(Resultado ,', ',Nom);
  /*se actualiza el nombre de registros que quedan por analizar*/
          SIETE cuántos=cuántos-1;
  TERMINAR MIENTRAS;
  CERRAR cursor1;
  RETORNO Resultado;

FIN $$
```

```
/*Probamos el funcionamiento*/
DELIMITADOR ;
SELECT ListaClientes('Madrid');
```

Ejemplo 1: Utilizando un HANDLER para controlar el bucle.

La variable salida está controlada por el HANDLER.

```
DELIMITADOR $$
CREATE FUNCTION ListadoClientes2(PCiudad VARCHAR(20))
DEVUELVE texto
COMENZAR

  DECLARO salida INT DEFAULT 0;
  DECLARAR Nombre VARCHAR(50) predeterminado '';
  DECLARAR Texto del resultado por defecto '';

  /*es declara el cursor para recorrer la consulta que obtiene los clientes*/
  DECLARAR cursor1 CURSOR PARA
  SELECT NombreCliente
  DE Clientes
  WHERE Ciudad=PCiudad;
  DECLARAR MANEJADOR CONTINUO PARA NO ENCONTRADO SET eixida=1;

  /*se abre el cursor*/
  ABIERTO cursor1;
  OBTENGA cursor1 EN SNombre;

  IF salida=0 then
              /*En el primero, no hay que meter como delante */
              SET Resultado =SNombre;
  FIN SI;

  /*se van concatenando los nombres mientras no llegamos al final del cursor*/
  WHILE salida=0 DO
              OBTENGA cursor1 EN SNombre;
              IF salida=0 THEN
                          SET Resultado = CONCAT(Resultado ,', ',SNombre);
              FIN SI;
  TERMINAR MIENTRAS;
  CERRAR cursor1;

  RETORNO Resultado;

FIN $$
```

```
/*Probamos el funcionamiento*/
DELIMITADOR ;
SELECT ListadoClientes2('Madrid');
```

Ejercicios

Manejo de errores en MySQL

1. Crea una BD llamada test que contenga una tabla llamada alumno. La mesa debe tener cuatro columnas:

- Id: entero sin signo (clave primaria).
- nombre: cadena de 50 caracteres.
- apellido1: cadena de 50 caracteres.
- apellido2: cadena de 50 caracteres.

Una vez creada la base de datos y la tabla deberá crear un procedimiento llamado insertar_alumno con las siguientes características. El procedimiento recibe cuatro parámetros de entrada (Id, nombre, apellido1, apellido2) y los insertará en la tabla alumno. El procedimiento devolverá como salida un parámetro llamado error que tendrá un valor igual a 0 si la operación ha podido realizarse con éxito y un valor igual a 1 en caso contrario.

Deberá manejar los errores que puedan ocurrir cuando se intenta insertar una fila que contiene una clave primaria repetida.

Transacciones con procedimientos almacenados

2. Escribe un procedimiento para introducir una Fila en Pedidos y otra en DetallePedidos.

Tienes que encapsular las sentencias críticas en una transacción de forma que si existe una SQLEXCEPTION, no se realice ninguna de las dos inserciones.

Los parámetros de entrada serán: CodigoCliente, CodigoProducto y Cantidad.

En la taula Pedidos:

- El CodigoPedido se obtendrá sumándole 1 al CodigoPedido máximo que haya en la tabla

- El DatePedido será la fecha del sistema.
- La FechaEsperada serà posterior en 5 dies a la FechaPedido.
- El CodigoCliente el del parámetro.
- El resto de campos en NULL

En la taula DetallePedidos:

- La Cantidad y el CodigoProducto los de los parámetros.
- El PrecioUnidad será el PrecioVenta del Código Producto correspondiente.
- La Línea será 1.

3. Crea un procedimiento que borre los datos de un empleado pasándole como parámetro un CodigoEmpleado. Para ello será necesario modificar primero el atributo CodigoJefe y ponerlo en NULL de los posibles subordinados del empleado en cuestión. Ambas operaciones estarán dentro de una transacción con un HANDLER que detecte cualquier problema.

El código asociado al HANDERL, además de hacer COMMIT, tendrá una SELECT que muestre un mensaje por ejemplo: 'No se puede eliminar el Empleado. Seguramente té Clientes que el referencien'

4. Crea una base de datos llamada cine que contenga dos tablas con las siguientes columnas.

Tabla cuentas:

- Id_cuenta: entero sin signo (clave primaria).
- equilibrio: signo de sentido real.

Mesa entradas:

- Id_butaca: ingrese sin signo (clave primaria).
- nif: cadena de 9 caracteres.

Una vez creada la base de datos y las tablas tendrá que crear un procedimiento llamado comprar_entrada con las siguientes características. El procedimiento recibe 3 parámetros de entrada (nif, Id_cuenta, Id_butaca) y tendrá un parámetro OUT

llamado error que tendrá un valor igual a 0 si la compra de la entrada se ha podido realizar con éxito y un valor igual a 1 de lo contrario.

El procedimiento de compra realiza los siguientes pasos:

- Inicia una transacción.
- Actualiza la columna saldo de la tabla cuentas cobrando 5 euros en la cuenta con el Id_cuenta adecuado.
- Inserta una fila en la tabla entradas indicando el sillón (Id_butaca) que acaba de comprar el usuario (nif).
- Comprueba si ha ocurrido algún error en las operaciones anteriores. Si no ocurre ningún error entonces aplica un COMMIT a la transacción y si ha ocurrido algún error aplica un ROLLBACK.

Deberá manejar los siguientes errores que puedan ocurrir durante el proceso.

- ERROR 1264 (Valor fuera de rango)
  - Se puede producir si al restar el saldo éste es negativo
- ERROR 1062 (Entrada duplicada para CLAVE PRINCIPAL)

¿Qué ocurre cuando intentamos comprar una entrada y le pasamos como parámetro uno número de cuenta que no existe en la tabla cuentas? Ocurre algún error o ¿podemos comprar la entrada?. Si es así (podemos comprar la entrada), propone una solución.

Cursores

Crea una función que devuelva un listado separado por comas con Nombre y Apellido1 de los empleados que tengan un determinado cargo (columna Puesto) pasado como parámetro.

Hazlo con controlando el cursor con un HANDLER y sin él.